Программирование на языке Java. Управление выполнением программы

Картузов А.В.

Управление в Java почти идентично средствам, используемым в C и C++.

1. Условные операторы

Они хорошо Вам знакомы, давайте познакомимся с каждым из них в Java.

1.1. if-else

В обобщенной форме этот оператор записывается следующим образом:

```
if (логическое выражение) оператор1; [ else оператор2;]
```

Раздел else необязателен. На месте любого из *операторов* может стоять *составной оператор,* заключенный в фигурные скобки. *Логическое выражение* — это любое выражение, возвращающее значение типа boolean.

```
int bytesAvailable;
// ...
if (bytesAvailable > 0) {
        ProcessData();
        bytesAvailable -= n;
} else
waitForMoreData();
```

А вот полная программа, в которой для определения, к какому времени года относится тот или иной месяц, используются операторы if-else.

```
class IfElse {
public static void main(String args[]) {
int month = 4;

String season;

if (month == 12 || month == 1 || month == 2) {
        season = "Winter";
} else if (month ==3 || month == 4 || month == 5) {
        season = "Spring";
} else if (month == 6 || month == 7 || month == 8) {
        season = "Summer";
} else if (month == 9 || month == 10 || month == 11) {
        season = "Autumn";
} else {
        season = "Bogus Month";
}
System.out.println( "April is in the " + season  + ".");

}
}
```

После выполнения программы вы должны получить следующий результат:

```
C: \> java IfElse

April is in the Spring.
```

1.2. break

В *языке* Java отсутствует оператор goto. Для того, чтобы в некоторых случаях заменять goto, в Java предусмотрен оператор break. Этот оператор сообщает исполняющей среде, что следует прекратить выполнение именованного блока и передать управление оператору, следующему за данным блоком. Для именования блоков в языке Java используются метки. Оператор break при работе с циклами и в операторах switch может использоваться без метки. В таком случае подразумевается выход из текущего блока.

Например, в следующей программе имеется три вложенных блока, и у каждого своя уникальная метка. Оператор break, стоящий во внутреннем блоке, вызывает переход на оператор, следующий за блоком b. При этом пропускаются два оператора println.

```
class Break {
public static void main(String args[]) { boolean t = true;

a:      { b:         { c:             {

System.out.println("Before the break"); // Перед break

       if (t)

       break b;

       System.out.println("This won't execute");  // Не будет  выполнено
   }

       System.out.println("This won't execute"); // Не будет  выполнено
   }

       System.out.println("This is after b"); //После b

} } }
```

В результате исполнения программы вы получите следующий результат:

```
C:\> Java Break

Before the break

This is after b
```

Замечание:

Вы можете использовать оператор break только для перехода за один из текущих вложенных блоков. Это отличает break от оператора goto языка C, для которого возможны переходы на произвольные метки.

1.3. switch

Оператор switch обеспечивает ясный способ переключения между различными частями программного кода в зависимости от значения одной переменной или выражения. Общая форма этого оператора такова:

```
switch ( выражение ) { case значение1:
        break;
        case значение2:
        break;
        case значением:
        break;
        default:
}
```

Результатом вычисления *выражения* может быть значение любого простого типа, при этом каждое из значений, указанных в операторах case, должно быть совместимо по типу с выражением в операторе switch. Все эти значения должны быть уникальными литералами. Если же вы укажете в двух операторах case одинаковые значения, транслятор выдаст сообщение об ошибке.

Если же значению выражения не соответствует ни один из операторов case, управление передается коду, расположенному после ключевого слова default. Отметим, что оператор default необязателен. В случае, когда ни один из операторов case не соответствует значению выражения и в switch отсутствует оператор default выполнение программы продолжается с оператора, следующего за оператором switch.

Внутри оператора switch (а также внутри циклических конструкций, но об этом — позже) break без метки приводит к передаче управления на код, стоящий после оператора switch. Если break отсутствует, после текущего раздела case будет выполняться следующий. Иногда бывает удобно иметь в операторе switch несколько смежных разделов case, не разделенных оператором break.

```
class SwitchSeason { public static void main(String args[]) {

int month = 4;

String season;

switch (month) {
```

```
case 12: // FALLSTHROUGH
case 1: // FALLSTHROUGH
case 2:
season = "Winter";
break;
case 3: // FALLSTHROUGH
case 4: // FALLSTHROUGH
case 5:
season = "Spring";
break;
case 6: // FALLSTHROUGH
case 7: // FALLSTHROUGH
case 8:
season = "Summer";
break;
case 9: // FALLSTHROUGH
case 10: // FALLSTHROUGH
case 11:
season = "Autumn";
break;

default:
season = "Bogus Month";

}

System.out.println("April is in the " +    season +     ".");

} }
```

Ниже приведен еще более полезный пример, где оператор switch используется для передачи управления в соответствии с различными кодами символов во входной строке. Программа подсчитывает число строк, слов и символов в текстовой строке.

```
class WordCount {

static String text = "Now is the tifne\n" +

                     "for all good men\n" +
```

```
                    "to come to the aid\n" +
                    "of their country\n"+
                    "and pay their due taxes\n";
static int len = text.length();
public static void main(String args[]) {
boolean inWord = false;
int numChars = 0;
int numWords = 0;
int numLines = 0;
for (int i=0; i < len; i++) {
     char c = text.charAt(i);
     numChars++;
     switch (c) {
             case '\n': numLines++; // FALLSTHROUGH
             case '\t': // FALLSTHROUGH
             case ' ' : if (inWord) {
                            numWords++;
                             inWord = false;
                        }
                        break;
```

```
            default: inWord = true;

                }

    }

System.out.println("\t" + numLines +"\t" + numWords + "\t" + numChars);

} }
```

В этой программе для подсчета слов использовано несколько концепций, относящихся к обработке строк. Подробно эти вопросы будут рассмотрены в главе 9.

1.4. return

В следующей главе вы узнаете, что в Java для реализации процедурного интерфейса к объектам классов используется разновидность подпрограмм, называемых методами. Подпрограмма main, которую мы использовали до сих пор — это статический метод соответствующего класса-примера. В любом месте программного кода метода можно поставить оператор return, который приведет к немедленному завершению работы и передаче управления коду, вызвавшему этот метод. Ниже приведен пример, иллюстрирующий использование оператора return для немедленного возврата управления, в данном случае — исполняющей среде Java.

```
class ReturnDemo {

public static void main(String args[]) {

boolean t = true;

System.out.println("Before the return"); //Перед оператором return

if (t) return;

System.out.println("This won't execute"); //Это не будет выполнено

} }
```

Замечание:

Зачем в этом примере использован оператор if (t)? Дело в том, не будь этого оператора, транслятор Java догадался бы, что последний оператор println никогда не будет выполнен. Такие случаи в Java считаются ошибками, поэтому без оператора if оттранслировать этот пример нам бы не удалось.

2. Циклы

Любой цикл можно разделить на 4 части — *инициализацию, тело, итерацию и условие завершения.* В Java есть три циклические конструкции: while (с пред-условием), do-while (с пост-условием) и for (с параметром).

2.1. while

Этот цикл многократно выполняется до тех пор, пока значение логического выражения равно true. Ниже приведена общая форма оператора while:

```
[ инициализация; ]
while ( завершение )     {
тело;
[итерация;] }
```

Инициализация и итерация необязательны. Ниже приведен пример цикла while для печати десяти строк «tick».

```
class WhileDemo {

public static void main(String args[]) {

int n = 10;

while (n > 0) {

      System.out.println("tick " + n);

      n—;

      }

} }
```

2.2. do-while

Иногда возникает потребность выполнить тело цикла по крайней мере один раз — даже в том случае, когда логическое выражение с самого начала принимает значение false. Для таких случаев в Java используется циклическая конструкция do-while. Ее общая форма записи такова:

```
[ инициализация; ] do { тело; [итерация;] } while ( завершение );
```

В следующем примере тело цикла выполняется до первой проверки условия завершения. Это позволяет совместить код итерации с условием завершения:

```
class DoWhile {

public static void main(String args[]) {

int n = 10;

do {

   System.out.println("tick " + n);

  } while (—n > 0);

} }
```

2.3. for

В этом операторе предусмотрены места для всех четырех частей цикла. Ниже приведена общая форма оператора записи for.

```
for ( инициализация; завершение; итерация ) тело;
```

Любой цикл, записанный с помощью оператора for, можно записать в виде цикла while, и наоборот. Если начальные условия таковы, что при входе в цикл условие завершения не выполнено, то операторы тела и итерации не выполняются ни одного раза. В каноническая форме цикла for происходит увеличение целого значения счетчика с минимального значения до определенного предела.

```
class ForDemo {

public static void main(String args[]) {

for (int i = 1; i <= 10; i++)

System.out.println("i = " + i);

} }
```

Следующий пример — вариант программы, ведущей обратный отсчет.

```
class ForTick {

public static void main(String args[]) {

for (int n = 10; n > 0; n—)

System.out.println("tick " + n);

} }
```

Обратите внимание — переменные можно объявлять внутри раздела инициализации оператора for. Переменная, объявленная внутри оператора for, действует в пределах этого оператора.

А вот — новая версия примера с временами года, в которой используется оператор for.

```
class Months {

static String months[] = {

"January", "February", "March",
"April", "May", "June",
"July", "August", "September",
"October", "November", "December" };

static int monthdays[]  =
{ 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };
```

```
static String spring = "spring";

static String summer = "summer";

static String autumn = "autumn";

static String winter = "winter";

static String seasons[]  =
{ winter, winter, spring, spring,
 spring, summer, summer, summer,
 autumn, autumn, autumn, winter };

public static void main(String args[]) {

for (int month = 0; month < 12; month++) {

System.out.println(months[month] + " is a "      +

seasons[month] + " month with " + monthdays[month] + " days.");

} } }
```

При выполнении эта программа выводит следующие строки:

```
C:\> Java Months

January is a winter month with 31 days.

February is a winter month with 28 days.

March is a spring month with 31 days.

April is a spring month with 30 days.

May is a spring month with 31 days.

June is a summer month with 30 days.

July is a summer month with 31 days.
```

```
August is a summer month with 31 days.

September is a autumn month with 30 days.

October is a autumn month with 31 days.

November is a autumn month with 30 days.

December a winter month with 31 days.
```

2.4. Оператор запятая

Иногда возникают ситуации, когда разделы инициализации или итерации цикла for требуют нескольких операторов. Поскольку составной оператор в фигурных скобках в заголовок цикла for вставлять нельзя, Java предоставляет альтернативный путь. Применение запятой (,) для разделения нескольких операторов допускается только внутри круглых скобок оператора for. Ниже приведен тривиальный пример цикла for, в котором в разделах инициализации и итерации стоит несколько операторов.

```
class Comma {

public static void main(String args[]) {

int a, b;

for (a = 1, b = 4; a < b; a++, b—) {

     System.out.println("a = " + a);

     System.out.println("b = " + b);

     }

} }
```

Вывод этой программы показывает, что цикл выполняется всего два раза.

```
C: \> java Comma
```

```
a = 1

b = 4

a = 2

b = 3
```

2.5. continue

В некоторых ситуациях возникает потребность досрочно перейти к выполнению следующей итерации, проигнорировав часть операторов тела цикла, еще не выполненных в текущей итерации. Для этой цели в Java предусмотрен оператор continue. Ниже приведен пример, в котором оператор continue используется для того, чтобы в каждой строке печатались два числа.

```
class ContinueDemo {

public static void main(String args[]) {

for (int i=0; i < 10; i++) {

     System.out.print(i + " ");

     if (i % 2 == 0) continue;

     System.out.println("");

     }

}}
```

Если индекс четный, цикл продолжается без вывода символа новой строки. Результат выполнения этой программы таков:

```
C: \> java ContinueDemo

0 1
```

```
2 3

4 5

5 7

8 9
```

Как и в случае оператора break, в операторе continue можно задавать метку, указывающую, в каком из вложенных циклов вы хотите досрочно прекратить выполнение текущей итерации. Для иллюстрации служит программа, использующая оператор continue с меткой для вывода треугольной таблицы умножения для чисел от 0 до 9:

```
class ContinueLabel {

public static void main(String args[]) {

outer:   for (int i=0; i < 10; i++) {

               for (int j = 0; j< 10; j++) {

                    if (j > i) {

                        System.out.println("");

                        continue outer;

                    }

               System.out.print(" " + (i * j));

               }

         }

}}
```

Оператор continue в этой программе приводит к завершению внутреннего цикла со счетчиком j и переходу к очередной итерации внешнего цикла со счетчиком i. В

процессе работы эта программа выводит следующие строки:

```
C:\> Java ContinueLabel

0

0 1

0 2 4

0 3 6 9

0 4 8 12 16

0 5 10 15 20 25

0 6 12 18 24 30 36

0 7 14 21 28 35 42 49

0 8 16 24 32 40 48 56 64

0 9 18 27 36 45 54 63 72 81
```

3. Исключения

Последний способ вызвать передачу управления при выполнении кода — использование встроенного в Java механизма обработки исключительных ситуаций. Для этой цели в языке предусмотрены операторы try, catch, throw и finally. Глава 10 целиком посвящена изучению механизма обработки исключительных ситуаций.

4. Вниз по течению

В последних четырех главах вы узнали о Java довольно много. Если бы это была книга по Фортрану, курс обучения можно было считать законченным. Однако по сравнению с Фортраном Java обладает дополнительными широкими возможностями. Поэтому давайте будем двигаться дальше и перейдем, наконец, к объектно-ориентированному

программированию.